MIDDLE EAST TECHNICAL UNIVERSITY

MIDDLE EAST TECHNICAL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

Vitriol

SOFTWARE REQUIREMENTS SPECIFICATIONS

GROUP MALLORN

Merve Bozo

Yaşar Berk Arı

Sertaç Kağan Aydın

Mustafa Orkun Acar

Team Leader: *Itır Önal*

Supervisor : *Asst.Prof.Dr. Pınar Karagöz*

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF FIGURES

TABLE OF SCREENSHOTS

1. INTRODUCTION

This Software Requirements Specification provides a complete description of all the functions and specifications of the Vitriol project, which is a generic machine learning tool targeting any domain and any user. This document includes purpose, features and the interfaces of the application. The intended audience of this document includes the prospective developers of the system. The software system to be produced is a generic, automated machine learning tool. The system mainly targeting software companies who lacks know how for big data. The system may be used by any company from any domain with no expertise. Any user from any company that knows the basics of computer usage will be able to use the system without any training.

During the preparation of the document, ISO-IEC-IEEE 29148-2011 is followed. External interfaces, functional requirements, use case diagrams, sequential diagrams, ER diagram and class diagrams are organized in the following sections.

1.1 PROBLEM DEFINITION

Big data is the one of the most trending topic in computer science nowadays, Moreover in modern world, information is the most valuable currency. The job of a data scientist is mainly to retrieve information from existing data. However it is a new topic and the sector lacks qualified scientist. Vitriol is aiming to solve that problem by eliminating the human factor from the data science in long term.

Vitriol is the first and only generic, automated machine learning tool that enables its users to run certain machine learning algorithms without any machine learning information. The system is developed in such a manner that, any person with basic computer skills can use the system without any extra information or training.

1.2 SYSTEM OVERVIEW

The system consist of 2 servers, 1 web application, 2 database, 1 memcached server and 1 machine learning machine. One of the servers is used for main application of the web service, the other one on the other hand, is used for notification mechanism, which is implemented via web socket technology. Web application is implemented mainly by html, JavaScript and css. The two databases are used for information storage. The first one is for the web application and the second one is for the data of the users. Memcached server is used for session storage. It is designed as a completely separate machine in order to commonize session objects between servers in case of the probability of using more than one server to balance the load of the server. Machine learning machine is the core of the system where all the algorithms and functionalities are applied.

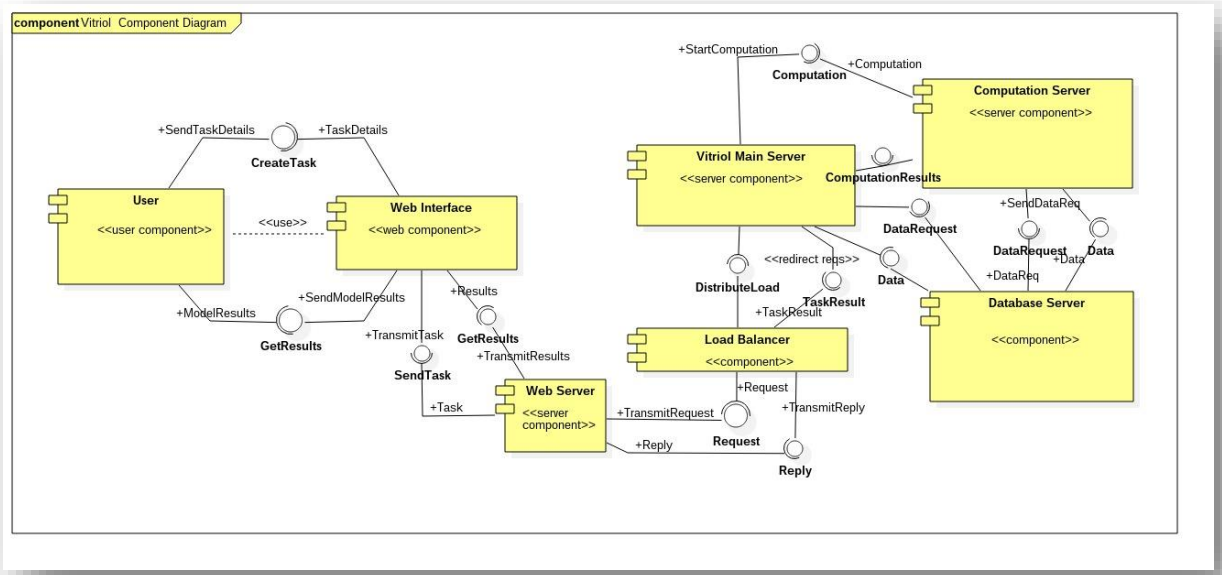


Figure 1: Component Diagram

The figure above is the component diagram for Vitriol. It shows the interactions between the system user, Web Interface, Main System Server, Database Server and the Computation Server. The main actions of the user are creating a new task on the system and viewing the reports for the previously started tasks. The user interacts with the system using the provided Web Interface. Web Interface is responsible for transmitting user requests to the server and also displaying the results returned to the user. All the requests arriving to the server side pass through the Load Balancer to minimize the response time and avoiding overloading a single system server. The main server handles the requests sent by the user by starting the desired operation and sending the results and obtained model to the user after it is completed. Main server basically communicates (sends the information about the operation) with the computation server to execute the operation. It also can send requests to the Database Server for some tasks such as authentication etc. Computation server is the fundamental hardware in which the algorithms are operated. It handles the requests by the main system server and generates models or performs other types of computations. To achieve this, it requests the required data from the database server. Upon completion, the obtained models and results are returned to the main server. Lastly, the database server is responsible for handling data requests by providing the required set of data to the Main Server or Computation Server.

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

SRS	Software requirements Specification
ML	Machine Learning
Npm	Node package manager
IEEE	Institute of Electrical and Electronics Engineers
DBMS	Database management system
ER Diagram	Entity Relationship Diagram
Member User	A user that registered to the system
MySQL	An open source database management system
TCP/IP	A communication protocol for the internet and similar networks
Notification	Instantaneous information that is shown to the users
Pm2	Process management system for nodejs
DB	Database

Table 1: Description of Definitions, Acronyms and Abbreviations

1.4 ASSUMPTIONS AND DEPENDENCIES

Both web servers are implemented assuming a Linux based operating system is running on the server machines (Centos [6] or Linux is preferable.) And also designs are made in such a manner that, server machines have java, scala, nodejs and spark installed in it already. If any of this assumptions will be changed, the design of the project and content of this document shall be changed accordingly.

The development team may assume that the users of the system has sql databases(Postresql, MySQL, mssql, oracle etc.) and have their data in a single table. In case of need of a join or limit operation, the user of the system is assumed to provide the table in already joined or limited state.

2. OVERALL DESCRIPTION

2.1 PRODUCT FUNCTIONS

The project Vitriol is a generic, automated machine learning tool that serve to any user with the data from any domain. It is designed as an on cloud service, hence the major parts for the project are web server and machine learning engine. The user that want to perform certain data operations on his/her data(such as anomaly detection or feature derivation) connects to the system via web interface, introduce his/her database by the help of the Ip (internet protocol) then choose what to do with the data(i.e. complete). That's all.

After that web service creates necessary records in the database and send signal to the ml engine to work on the data. As soon as the ml engine finishes the job, it inserts the necessary information to the notification table in the database. When the data is inserted a trigger function works on the database which invokes a web socket server, and web socket server pushes the notification in to the client side of the web system.

2.1.1 USE-CASE MODEL SURVEY

The system works for only one user (namely user) which is able to work every aspect of the project. In future development it is being considered to have different type of users for different type of action, however it is out of scope of this document. The use cases, and detailed explanations of the use cases are demonstrated below.

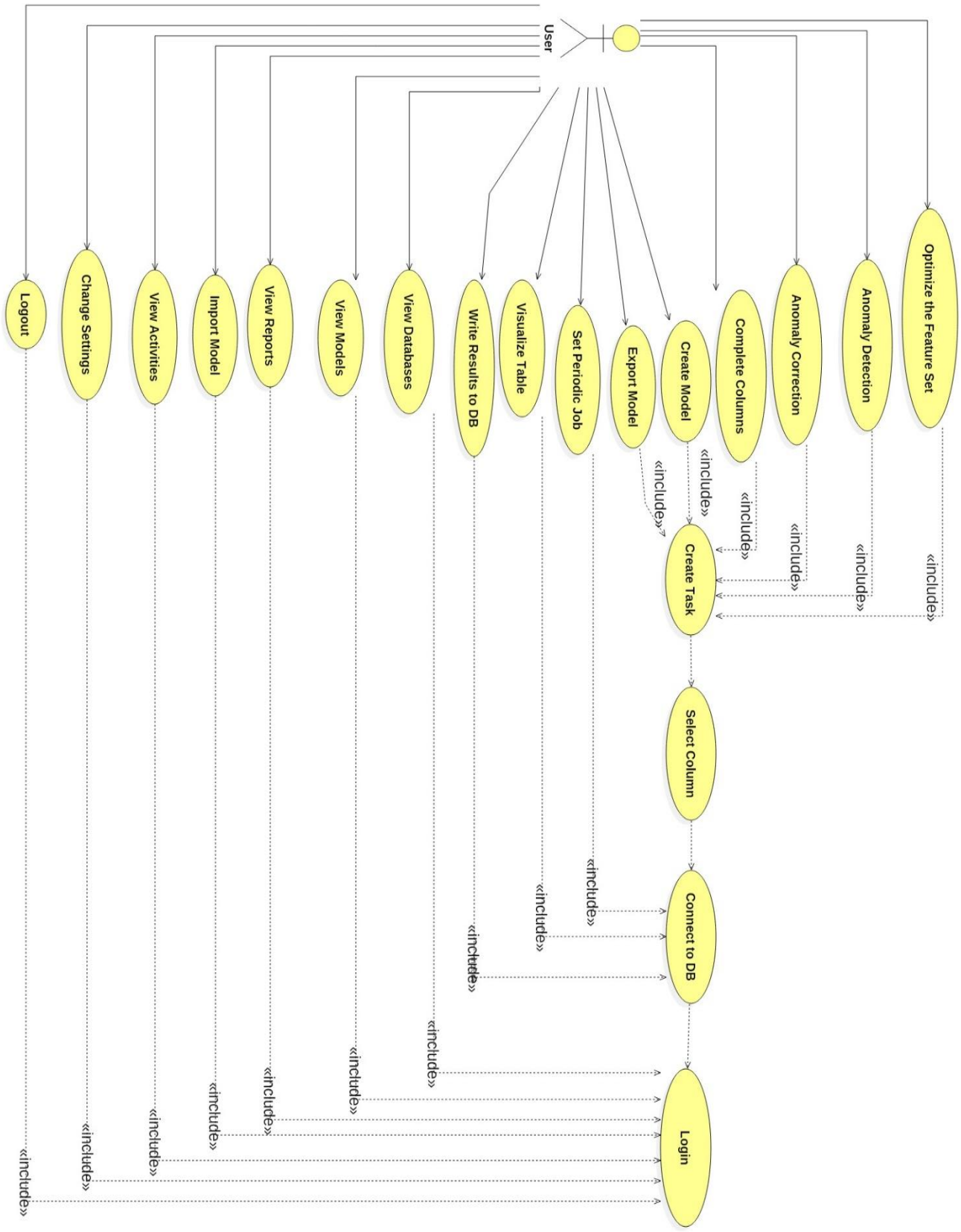


Figure 2: General Use Case Diagram

No	Functionality	Short Description
1	Login	Logging into system with membership and password
2	Connect to Database	Connect a remote database
3	View Databases	View all databases
4	Create Task	Creation of a task for processing the data
5	Select Column(s) from table	Select specific columns for starting process
6	Anomaly Detection	Show irrelevant parts of the data
7	Anomaly Correction	Correct irrelevant parts of table
8	Create Model for selected column	Create a model on selected columns for future use
9	Complete Selected Column	Complete empty parts of selected column
10	Optimize a feature set of selected column(s)	Create new features using selected column(s)
11	Import Model	Import an external model
12	Export Model	Export a model for future use
13	View Models	View previously created models
14	View Reports	View the detailed reports about process
15	Visualize Table	Visualize the distribution of data points in a table
16	Set Periodic Job	Set a specific job to work periodically
17	View Activities	Logs of previous activities
18	Change Settings	Change user settings (password, e-mail, username etc.)
19	Logout	Logout from system

Table 2: Overview of the Use-Case

Vitriol system provides many functionalities as seen above. All of them requires user to be logged in, which is also a use case for the user. Some of the cases include other cases also. For sake of simplicity each use case is explained separately below.

2.1.1.1 LOGIN USE CASE FOR USER

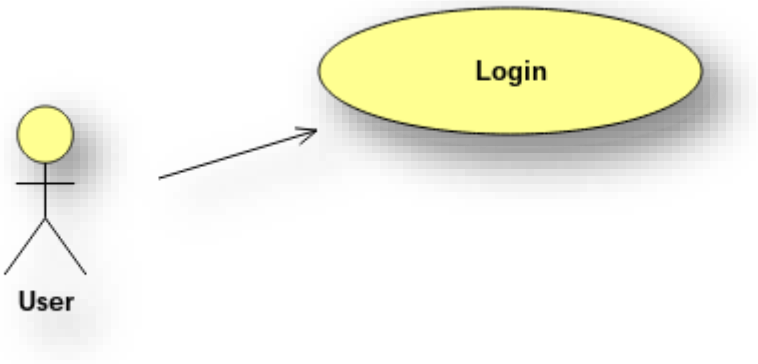


Figure 3: Login Use Case

Use Case ID	UC1
Use Case Name	Login
Description	This use case describe event in which a user login to system with his/her username and password.
Actors	User
Precondition	-
Trigger	The user tries to login to the system using the web service providing login credentials.
Basic Flow	1- The user enters his/her username 2- The user enters his/her 8 digit password 3- Web service checks the database for password confirmation via server 4- Web service displays a correct password message
Exception Flow	If the entered password does not match with the password in the database an error message is displayed by web service.
Post Conditions	-

Table 3: Description of Login Use Case

2.1.1.2 CONNECT TO DATABASE USE CASE FOR USER

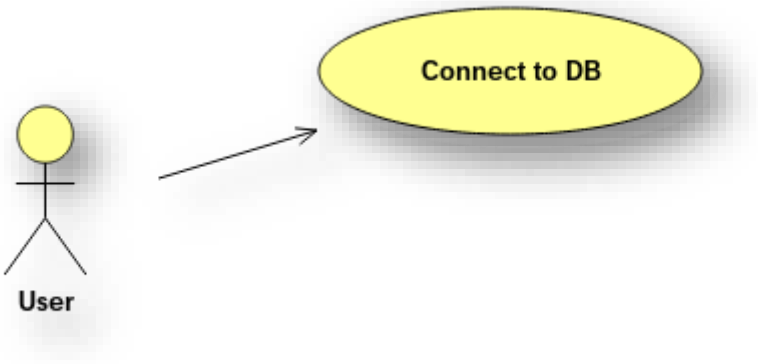


Figure 4: Connect to Databases Use Case

Use Case ID	UC2
Use Case Name	Connect to Database
Description	This use case describe event in which a user connect to database with its credentials
Actors	User
Precondition	-
Trigger	The user tries to connect a DB to his Vitriol account by providing the DB credentials such as DB address and password.
Basic Flow	1- The user enters the remote database Ip address 2- The user enters his/her database password 3- If the password is accepted by remote database, database table names are shown
Exception Flow	If the password is refused by remote database, an error message is displayed by web service
Post Conditions	-

Table 4: Description of Connect to Databases

2.1.1.3 DETECT ANOMALY USE CASE FOR USER

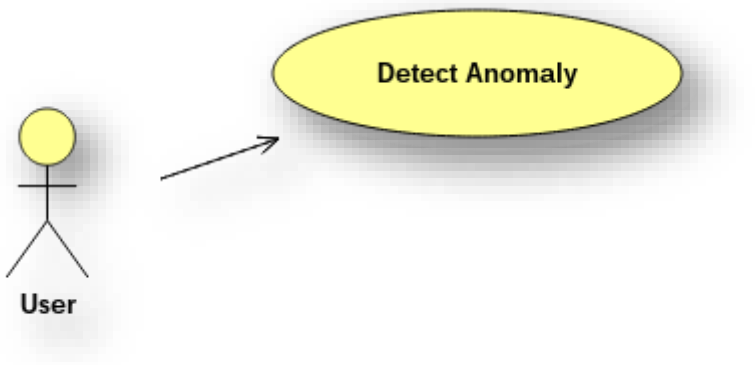


Figure 5: Detect Anomaly Use Case

User Case ID	UC3
User Case Name	Detect Anomaly
Description	This use case describes the operation that user can start a task to detect different kinds of anomalies represented in the dataset.
Actors	Member User
Precondition	User has to login the system, connect to a database and creates the task after selecting some columns.
Trigger	User creates anomaly detection task using the interface.
Basic Flow	1 - Anomaly detection module operates on the selected columns. 2 - Results are shown on the reports page.
Exception Flow	-
Post Conditions	Data sets are analyzed and anomalies detected to be shown.

Table 5: Description of Detect Anomaly Use Case

2.1.1.4 CORRECT ANOMALY USE CASE FOR USER

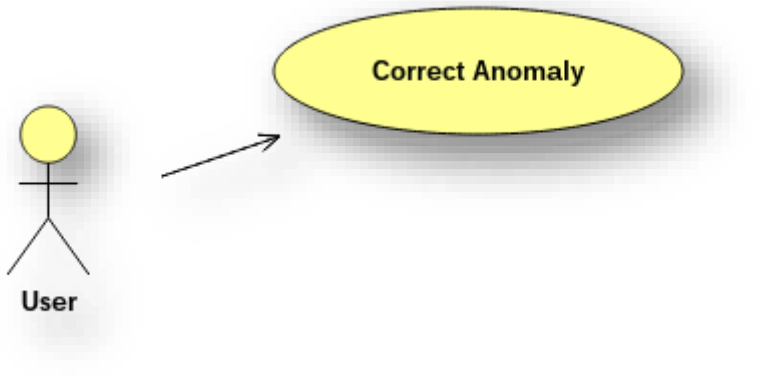


Figure 6: Correct Anomaly Use Case

User Case ID	UC4
User Case Name	Correct Anomaly
Description	This use case describes the operation in which inconsistent data points are cleared or updated.
Actors	Member User
Precondition	User has to login to the system, connect to a database and creates the task after selecting the columns.
Trigger	User creates anomaly correction task.
Basic Flow	1 - Anatoly correction module operates on the selected columns. 2 - Datasets are updated in databases. 3 - Results are shown on the reports page.
Exception Flow	-
Post Conditions	Datasets are analyzed and anomalies detected to be shown and datasets are updated according to correction or deletion decisions.

Table 6 Description of Correct Anomaly Use Case

2.1.1.5 CHANGE SETTINGS USE CASE FOR USER

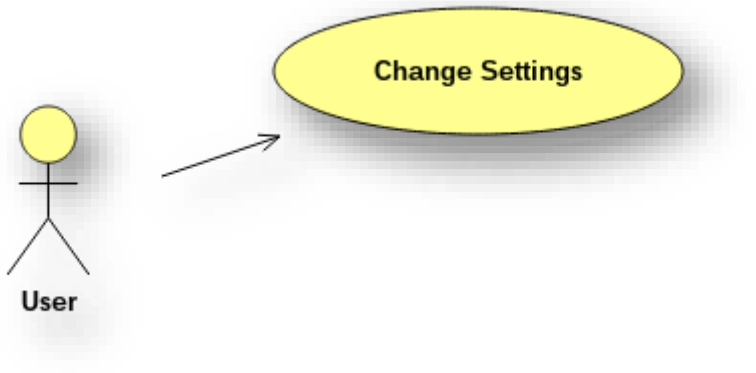


Figure 7: Change Settings Use Case

User Case ID	UC5
User Case Name	Change Settings
Description	This use case describes that user can change personal settings such as passwords, avatars, security questions etc.
Actors	Member User
Precondition	User must be logged in to the system.
Trigger	User clicks on the 'change settings' button.
Basic Flow	1-User selects the data that he/she wants to update. 2-User enters the new value. 3 - New values are checked according to some expected range. If it is valid, database entry is updated accordingly.
Exception Flow	1 - If the new entered value is not in the expected range, an exception occurs and an error message are shown to the user.
Post Conditions	Updated user information in the database.

Table 7: Description of Change Settings Use Case

2.1.1.6 COMPLETE COLUMNS USE CASE FOR USER

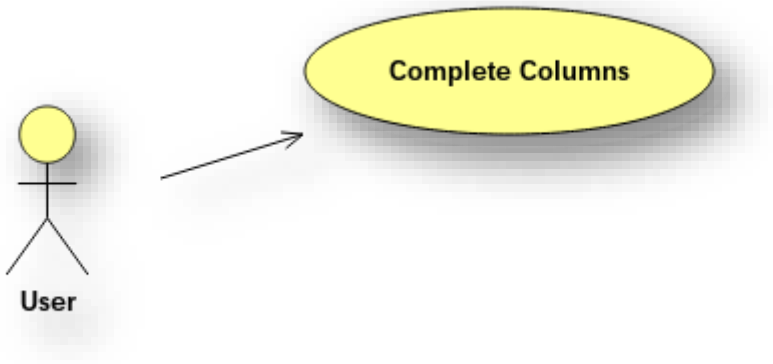


Figure 8: Complete Columns Use Case

User Case ID	UC6
User Case Name	Complete Columns
Description	This use case describes the operation of the completion of missing and invalid entries in columns of the databases.
Actors	Member User
Precondition	User must be logged in to the system and have at least one connected database.
Trigger	User starts the data completion task from the interface.
Basic Flow	1 - User select the column that operation will be applied on. 2 - Complete columns module works on the selected column and completes the missing values. 3 - If the multiple columns are selected by the user, the operation is performed on the selected columns consecutively.
Exception Flow	-
Post Conditions	Updated datasets are shown to the user. A report is generated about the task.

Table 8: Description of Complete Columns Use Case

2.1.1.7 WRITE RESULTS TO DB USE CASE FOR USER

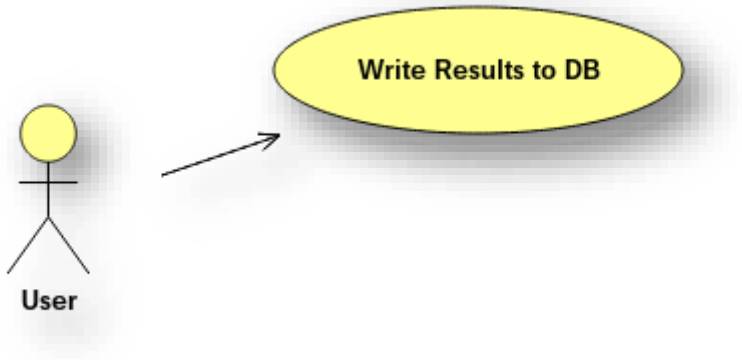


Figure 9: Write Results to DB Use Case

User Case ID	UC7
User Case Name	Write Results to DB
Description	This use case describes the operation of write the obtained results after operations to the database of the user
Actors	Member User
Precondition	User must be logged in to the system and have at least one connected database. User must create task and the execution of the task should be completed.
Trigger	User clicks on the 'write to database' button on the web interface.
Basic Flow	1 – User clicks on the 'write to database' column on the web interface. 2 – A connection between the web server and the database of the user is established. 3 – Data is copied to the database of the user.
Exception Flow	-
Post Conditions	-

2.1.1.8 CREATE TASK USE CASE FOR USER

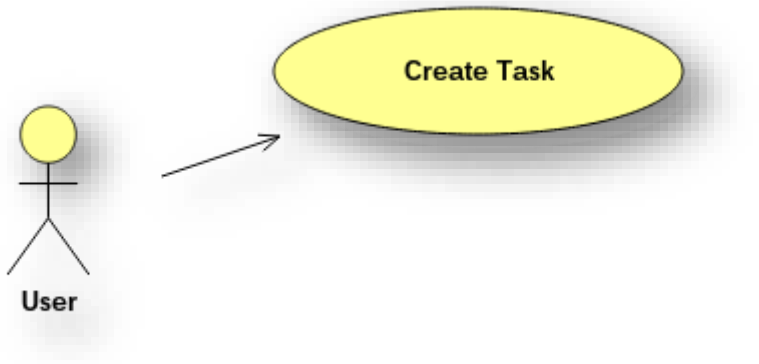


Figure 10: Create Task Use Case

User Case ID	UC8
User Case Name	Create Task
Description	This use case describes the operation that allows the user to specify the details of the desired task.
Actors	Member User
Precondition	User must be logged in to the system
Trigger	User clicks on the create model button.
Basic Flow	1-The new window is shown to the user asking for the details about the operation that will be created.
Exception Flow	1 - If the user didn't provide columns of a table, exception occurs and an error massage are shown to the user.
Post Conditions	New task added to the system job queue having user specified type.

Table 9: Description of Create Task Use Case

2.1.1.9 OPTIMIZE FEATURE SET USE CASE FOR USER

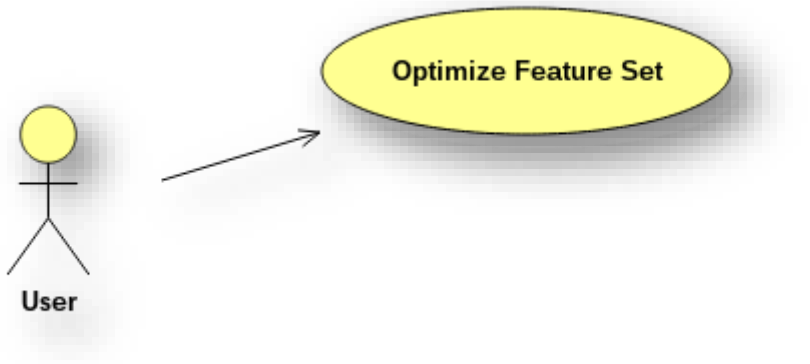


Figure 11: Optimize Feature Set Use Case

User Case ID	UC9
User Case Name	Optimize Feature Set
Description	This use case describes the operation that optimizes the feature set.
Actors	Member User
Precondition	User must be logged in to the system and created the related task.
Trigger	User selects the 'optimize the feature set' operation in task creation phase.
Basic Flow	1 - Related task is performed on the dataset provided by the user. 2 - The result are shown on the report page.
Exception Flow	-
Post Conditions	Feature sets are optimized.

Table 10: Description of Optimize Feature Set Use Case

2.1.1.10 EXPORT MODEL USE CASE FOR USER

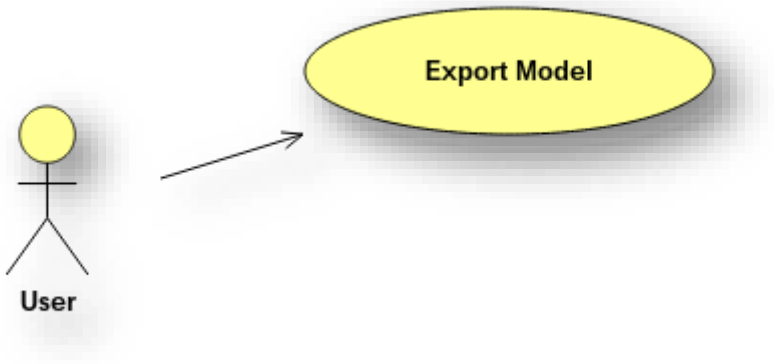


Figure 12: Export Model Use Case

User Case ID	UC10
User Case Name	Export Model
Description	This use case describes the operation of creating xml encoding of an existing model created by the system.
Actors	Member User
Precondition	User must be logged in the system.
Trigger	User selects a model and clicks the 'export model' button.
Basic Flow	1-Existing model encoded as xml file. 2-Encoded file is downloaded by the user.
Exception Flow	-
Post Conditions	Xml file created.

Table 11: Description of Export Model

2.1.1.11 IMPORT MODEL USE CASE FOR USER

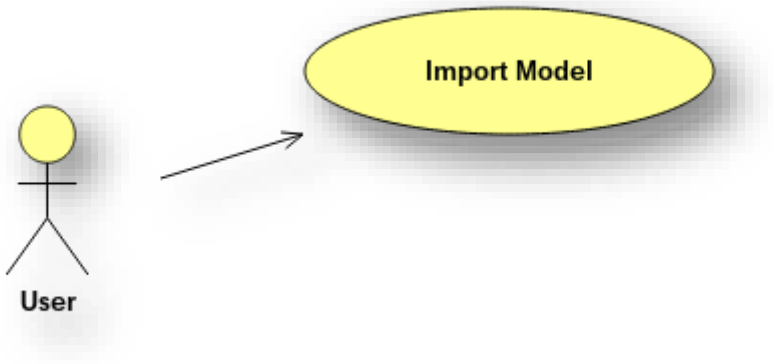


Figure 13: Import Model Use Case

User Case ID	UC11
User Case Name	Import Model
Description	This use case describes the operation of importing XML encoded model and addition to the user's models.
Actors	Member User
Precondition	User must be logged in to the system.
Trigger	User selects a model and clicks the 'import model' button.
Basic Flow	1 – User selects a model file to import. 2 – Selected file is checked for validity and if successful, inserted into the user's models.
Exception Flow	1 – If the selected file for import is invalid, an error message is shown to the user.
Post Conditions	An external method is imported and added to the user's models.

Table 12: Description of Import Model Use Case

2.1.1.12 SELECT COLUMN USE CASE FOR USER

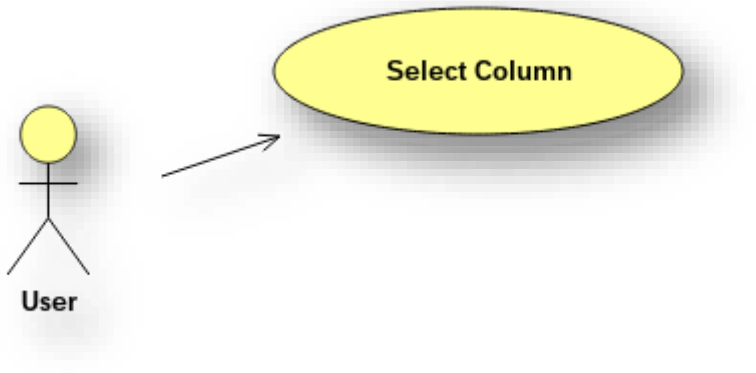


Figure 14: Select Column Use Case

User Case ID	UC12
User Case Name	Select Columns
Description	This use case describes an event in which a user selects the columns that will be processed.
Actors	User
Precondition	The user has to login the system and connect to a database.
Trigger	The user clicks the “Select Columns” button.
Basic Flow	1 – The user selects the desired columns by clicking on them on a table preview screen. 2 – Selection process ends when the user clicks “OK” button.
Exception Flow	-
Post Conditions	Some columns are selected by the user for the process.

Table 13: Description of Select Columns Use Case

2.1.1.13 SET PERIODIC JOB USE CASE FOR USER

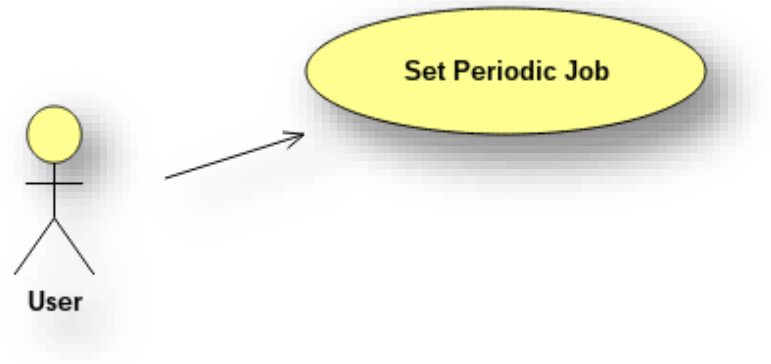


Figure 15: Set Periodic Job Use Case

User Case ID	UC13
User Case Name	Set Periodic Job
Description	This use case describes the operation of setting a job to be run in the future, possibly periodically.
Actors	Member User
Precondition	The user must be logged in to the system.
Trigger	User chooses the operation using the interface.
Basic Flow	1 – User chooses the operation by clicking the button in 'Jobs' tab. 2 – User creates a periodic job by providing the job (operations) and timing information and confirms by clicking 'OK'.
Exception Flow	1 – If the user clicks 'OK' button without providing an operation as s/he creates the periodic job, an error message is displayed.
Post Conditions	A periodic job to be run in a future time is added to the job queue.

Table 14: Description of Set Periodic Job Use Case

2.1.1.14 VIEW ACTIVITIES USE CASE FOR USER

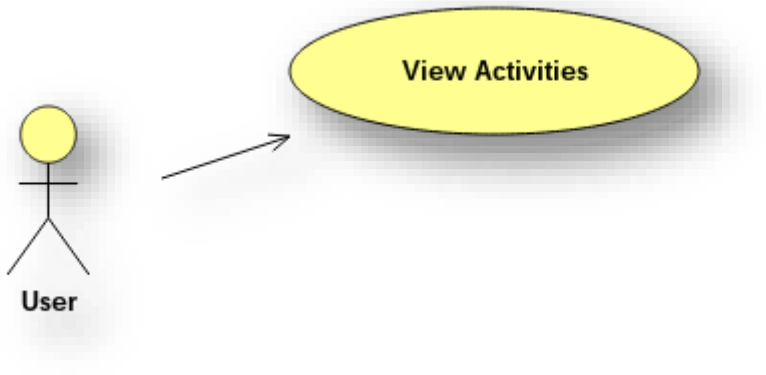


Figure 16: View Activities Use Case

User Case ID	UC14
User Case Name	View Activities
Description	This use case describes the operation of viewing past activities by that account.
Actors	Member User
Precondition	The must be logged in to the system.
Trigger	User clicks on the 'View Activities ‘tab.
Basic Flow	1 - User chooses the action. 2 – Past activities are displayed. If there is no activity, a message is displayed instead.
Exception Flow	-
Post Conditions	The past activities of the user is displayed.

Table 15: Description of View Activities Use Case

2.1.1.15 VIEW DATABASES USE CASE FOR USER

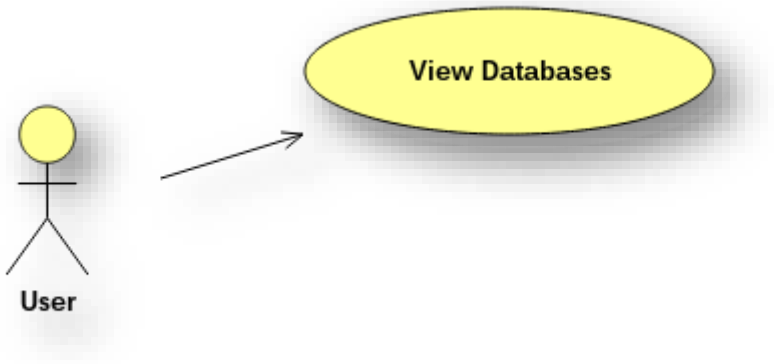


Figure 17: View Databases Use Case

User Case ID	UC15
User Case Name	View Databases
Description	This use case describes the operation of viewing databases connected to the system.
Actors	Member User
Precondition	The user must be logged in to the system.
Trigger	User clicks on the 'View Databases' tab.
Basic Flow	1 - User chooses the action. 2 – Connected databases are displayed. If there is no databases, a message is displayed instead.
Exception Flow	-
Post Conditions	The connected databases are displayed.

Table 16: Description of View Databases Use Case

2.1.1.16 VIEW MODELS USE CASE FOR USER

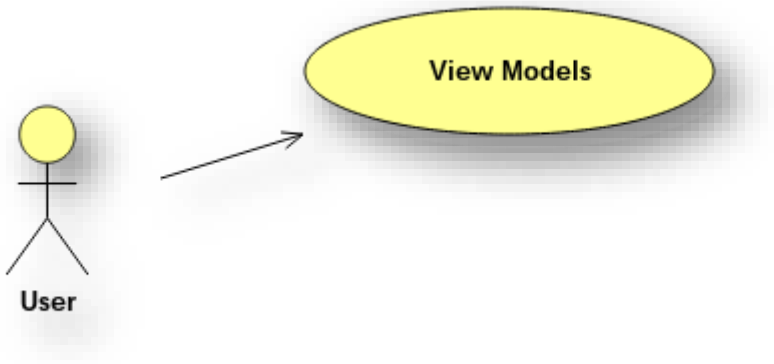


Figure 18: View Models Use Case

User Case ID	UC16
User Case Name	View Models
Description	This use case describes the operation of viewing previously created models.
Actors	Member User
Precondition	The user must be logged in to the system.
Trigger	User clicks on the 'View Models' tab.
Basic Flow	1 - User chooses the action. 2 – Models are displayed. If there is no models, a message is displayed instead.
Exception Flow	-
Post Conditions	The models are shown.

Table 17: Description of View Models Use Case

2.1.1.17 VIEW REPORTS USE CASE FOR USER

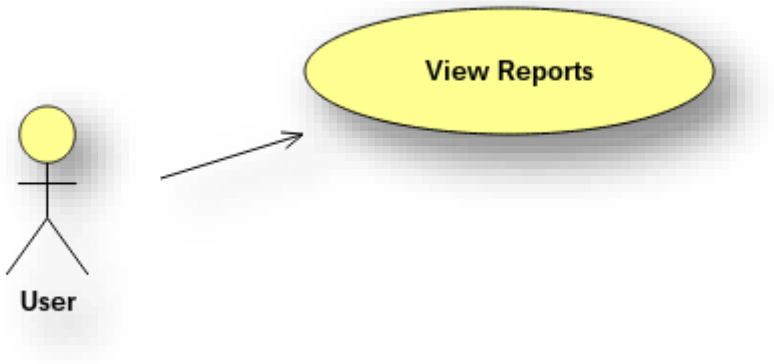


Figure 19: View Reports Use Case

User Case ID	UC17
User Case Name	View Reports
Description	This use case describes the operation of viewing created reports about previous processing of the user.
Actors	Member User
Precondition	The user must be logged in to the system.
Trigger	User clicks on the 'View Reports' tab.
Basic Flow	1 - User chooses the action. 2 – Reports about the past actions are displayed. If there is no reports, a message is displayed instead.
Exception Flow	-
Post Conditions	The reports are shown.

Table 18: Description of View Reports Use Case

2.1.1.18 VISUALIZE USE CASE FOR USER

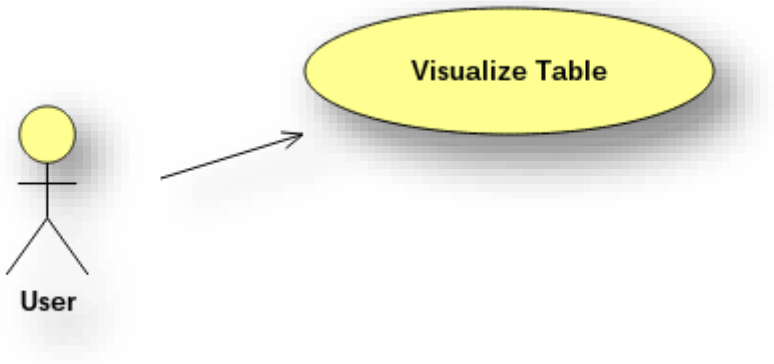


Figure 20: Visualize Table Use Case

User Case ID	UC18
User Case Name	Visualize Table
Description	This use case describes the operation in which a database table is visualized for better understanding.
Actors	Member User
Precondition	User must be logged in to the system. At least one table must be connected to user's account.
Trigger	User clicks on the 'Visualize table' button after selecting a table.
Basic Flow	1 – User selects a table and chooses visualizing operation with options of 2D and 3D visualization. 2 – The related algorithm produces the resulting graph and it is shown to the user.
Exception Flow	-
Post Conditions	The desired visualization is obtained.

Table 19: Description of Visualize Table Use Case

2.1.1.19 LOGOUT USE CASE FOR USER

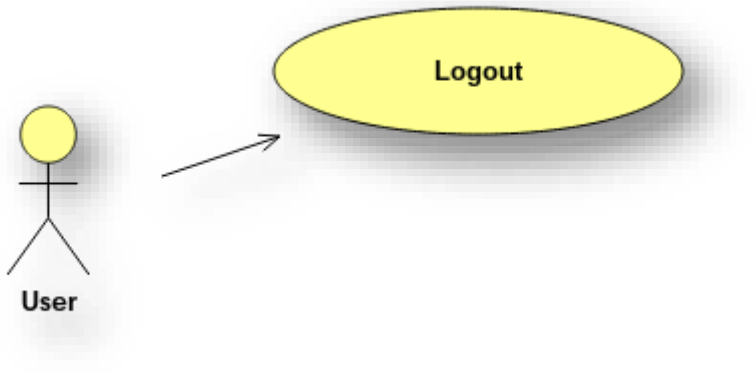


Figure 21: Logout Use Case

User Case ID	UC19
User Case Name	Logout
Description	This use case describes an event in which a user logs out from the system.
Actors	User
Precondition	User has to login the system
Trigger	The user tries to log out from the system using log out button.
Basic Flow	1 - User clicks the log out button 2 - Session is finished by the web service
Exception Flow	-
Post Conditions	The user is redirected to the login screen

Table 20: Description of Logout Use Case

2.1.2 ACTOR SURVEY

The vitriol system consists of only one actor type namely user or member user. The details of the actor can be seen in the table below.

Actor Name	Member User
Description	Member user or shortly the user is anyone that registers and get a password from Vitriol system. All the functionalities of Vitriol can be used by member user.

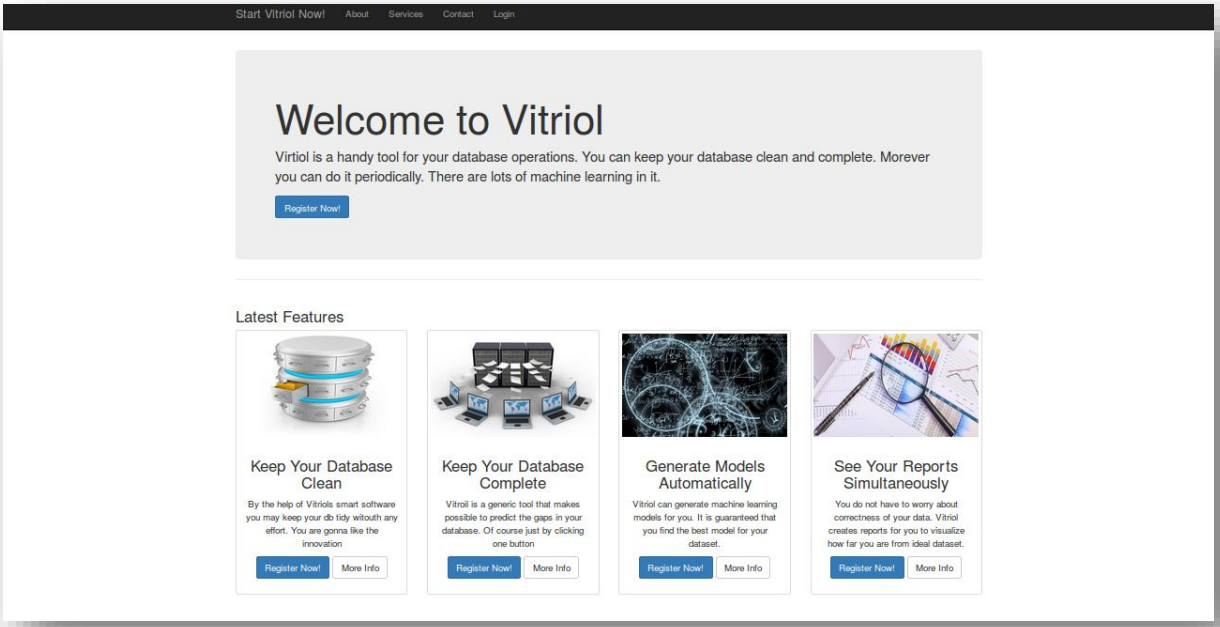
Table 21: Description of Actor Survey

2.2 INTERFACES

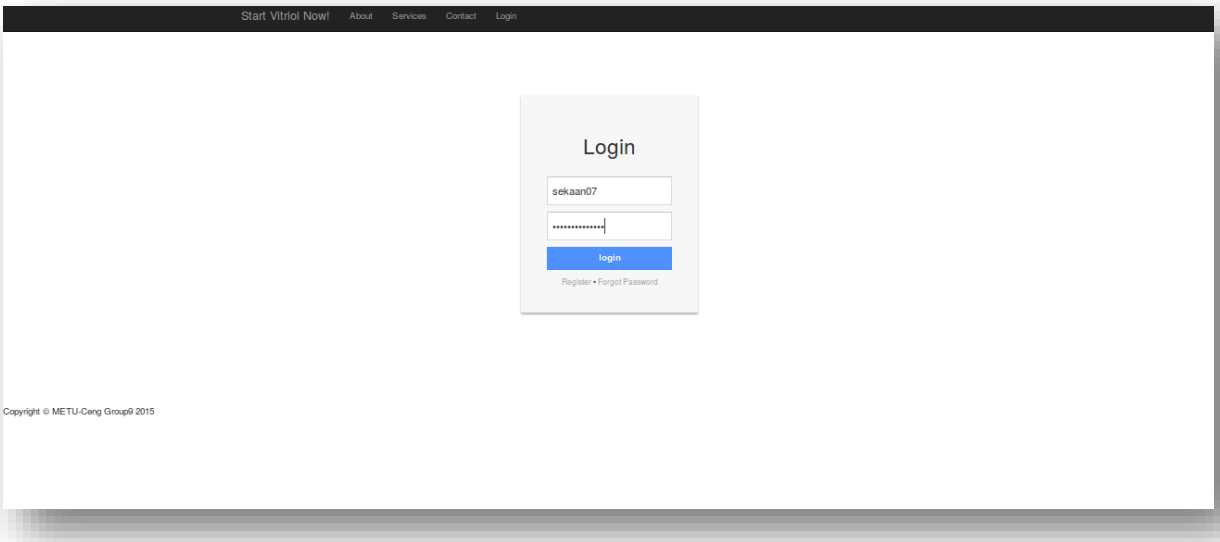
2.2.1 USER INTERFACES

The user interface of the web application is comprehensive and easy to use. At the header part user is able to see his/her notifications and manage the setting of his/her account. In the left-most part of the page there is a list of operation that user can do. This operations are mainly the use cases that described in section 2.1.1. This two sections are fixed in every page. At the middle, beside from these sections, there is a white board which shows a little variations regarding of the operation that user want to do. For example if the user wants to connect database, the left-most and the top section is the same but in the white board he/she sees a form to connect database. If he/she wants to see the reports some charts is demonstrated instead.

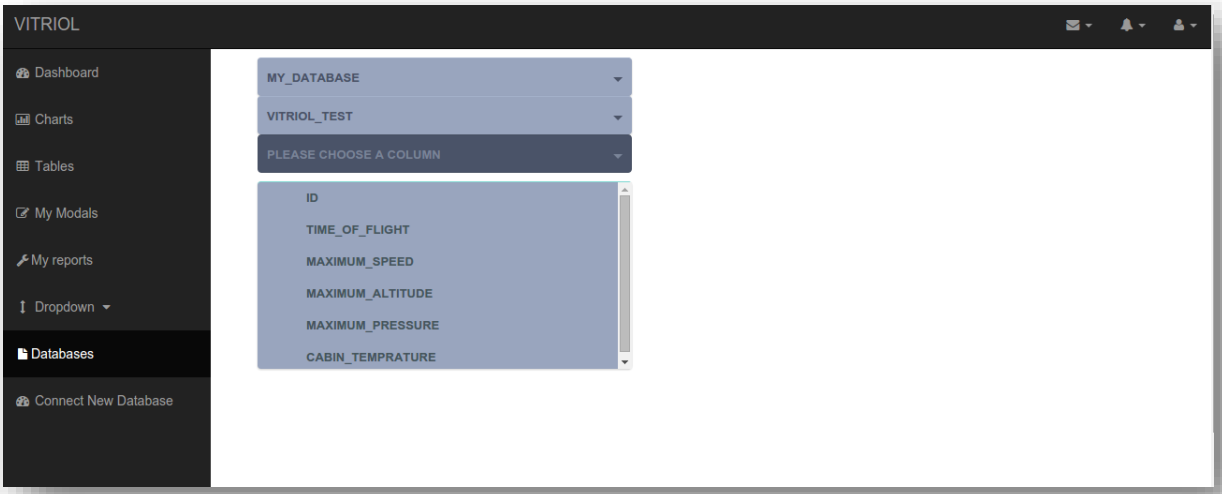
The details of the user interface can be observed by looking at the figures below.



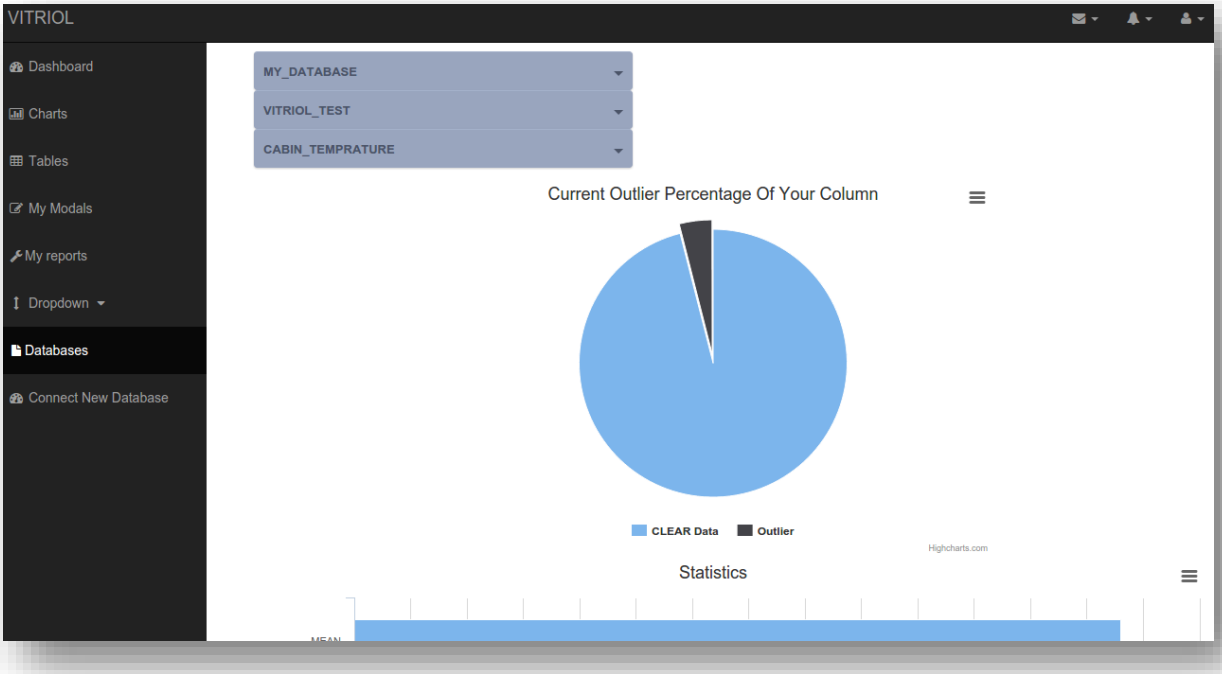
Screenshot 1: Home Page of Vitriol



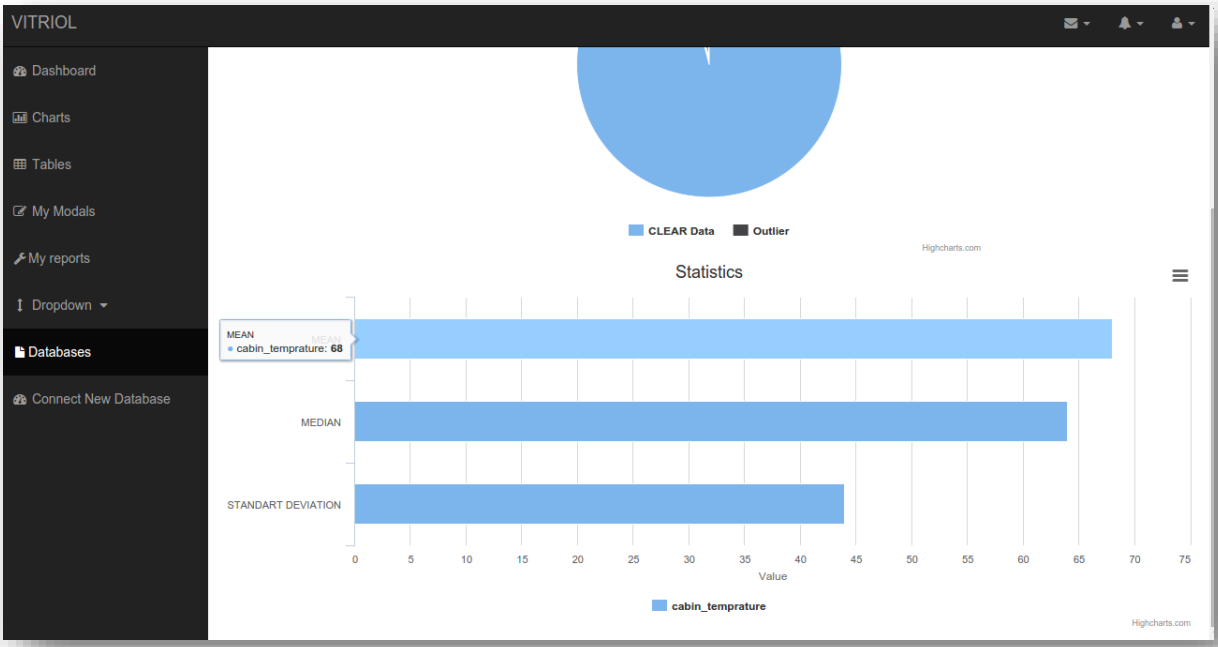
Screenshot 2: Login Page of Vitriol



Screenshot 3: Choose Column Page of Vitriol



Screenshot 4: Pie Chart of Results



Screenshot 5: Statistical Analyze of Results

2.2.2 HARDWARE INTERFACES

Since it is an on cloud web service, the project requires a server machine (physical or virtual), which shall be reliable. SATA and SAS disk bus standards will be adopted along with RAID technology so that, better maintenance and reliability of the persistence storage can be achieved. And also CPU and Memory of the server shall be as high as possible (8 GB RAM, Intel Xeon e5-2699 v3 @ 2.30 GHz at least) to achieve %99.99 availability constraint.

If more than one server is going to be used behind a load balancer, than another machine for in-memory caching memory would also be beneficial in order to commonize session objects.

2.2.3 SOFTWARE INTERFACES

- The server machine shall have a Linux environment namely Ubuntu or CentOs version at least 14.04 and 6.7 accordingly
- Java Runtime Environment (Java RE), version 1.6 or later, is required for ml engine, which is used on the server side of the product.
- Apache Spark version 1.6 is required by the ml engine
- Nodejs environment and it package manager npm shall be installed on the server.

- Memcached is used for session storage. Hence it must either be used as a distinct machine or the server that uses memcached shall have memcached installed on it.
- Postgresql version 9 or higher is used for the database server.
- Apache tomcat will be used for a java bridge server for the communication of web server and ml engine.

2.2.4 COMMUNICATIONS INTERFACES

The server and Database Management System communicate using the TCP/IP protocol. And also the notification mechanism is designed by using web socket protocol.

2.3 CONSTRAINTS

The one and only constraint for the design is to not to work on data on the database of the user. Since every query that is being worked on the database results in a cost, the project have to replicate the data to its own database to work on it.

2.4 ASSUMPTIONS AND DEPENDENCIES

- The development team may assume that, the data that is being the object of the project will be provided as a single sql table. If any join or limitation will be performed on the data, that is assumed to be done by the end user.
- Another assumption is that the database of the user is open to remote access.
- The last assumption can be made by the developers is that the use will connect a sql database, which fulfills the basic requirements of sql tables.

3. SPECIFIC REQUIREMENTS

In this section, all of the software requirements specified in details so that designers can design the system and testers can test to the system to satisfy these requirements.

3.1 FUNCTIONAL REQUIREMENTS

3.1.1 FUNCTIONAL REQUIREMENT 1 – LOGIN

The user shall be log into system by a username and password. Password should be kept in the database in an encrypted version for security concerns. The login mechanism shall work in cooperate with session mechanism. Whenever a user logs in a session shall be created and shall be kept in in-cache memory and browser cookie. If another login occurs for the same user name the previous session shall be dropped. And also if the user that logs into system shows no action for 15 minutes the session again shall be dropped. The login information shall be logged into the database for the sake of user reports.

3.1.2 FUNCTIONAL REQUIREMENT 2 – CONNECT DATABASE

The user shall be able to connect his/her database just by providing the credentials such as Ip address, port number username and password of the database. No other information shall be asked to the user.

3.1.3 FUNCTIONAL REQUIREMENT 3 - DETECT ANOMALIES

The user shall be able to detect anomalies of the data from the database that he/she connected. This is the process of viewing irrelevant parts of the data.

3.1.4 FUNCTIONAL REQUIREMENT 4 - CORRECT ANOMALIES

The user shall be able to correct the anomalies that he/she detects. This process includes the functional requirement 3.

3.1.5 FUNCTIONAL REQUIREMENT 5 – CHANGE SETTINGS

The user shall be able to change his/her setting such as password or username.

3.1.6 FUNCTIONAL REQUIREMENT 6 – COMPLETE SELECTED COLUMN

The user shall be able to complete the data in the selected column if it has missing parts. This completion shall be done by at least %60 accuracy. This process includes functional requirement 12.

3.1.7 FUNCTIONAL REQUIREMENT 7 – WRITE RESULTS TO DB

The user shall be able to write results of his/her tasks to database.

3.1.8 FUNCTIONAL REQUIREMENT 8 - CREATE TASK

The user shall be able to create task in order to process the data. The created task shall be remembered by the system.

3.1.9 FUNCTIONAL REQUIREMENT 9 – OPTIMIZE A FEATURE SET OF SELECTED COLUMN(S)

The user shall be able to optimize a feature set for the selected column or columns.

3.1.10 FUNCTIONAL REQUIREMENT 10 – EXPORT MODEL

The user shall be able to export the ml model that is generated for his/her data in order to use it in the system in the future.

3.1.11 FUNCTIONAL REQUIREMENT 11 – IMPORT MODEL

The user shall be able to import ml model that is exported before or created by himself/herself.

3.1.12 FUNCTIONAL REQUIREMENT 12 - SELECT COLUMN(S) FOR THE TABLE

The user shall be able to select column(s) from the table that is inside the database that he/she connected. The columns shall be listed to the user for him/her to choose.

3.1.13 FUNCTIONAL REQUIREMENT 13 – SET PERIODIC JOB

The user shall be able to set periodic jobs for any task that he/she wants. The periodicity of the task may be weekly, two times a month, monthly and never.

3.1.14 FUNCTIONAL REQUIREMENT 14 – VIEW ACTIVITIES

The user shall be able to see previous activities taken by himself/herself in order to keep track on the process of his/her data.

3.1.15 FUNCTIONAL REQUIREMENT 15 - VIEW DATABASES

The user shall be able to see his/her databases that are connected before. The databases shall be listed for the user to choose.

3.1.16 FUNCTIONAL REQUIREMENT 16 – VIEW MODELS

The user shall be able to see the ml models that are created before for his/her data. In other words system shall remember the models that is created before.

3.1.17 FUNCTIONAL REQUIREMENT 17 – VIEW REPORTS

The user shall be able to view reports about his/her process.

3.1.18 FUNCTIONAL REQUIREMENT 18 – VISUALISE TLABLE

The user shall be able to see the distribution of the data points in his/her table. This visualization shall be done by using dot plots.

3.1.19 FUNCTIONAL REQUIREMENT 19 – LOG OUT

The user shall be able to log out and kill his/her session whenever he/she wants.

3.2 NONFUNCTIONAL REQUIREMENTS

3.2.1 USABILITY

Vitriol is a handy tool that requires no extra training. It is targeting mainly software companies but this is not a requirement. Any user that has only basic computer knowledge will be able to use the system easily.

3.2.2 RELIABILITY

If any component of the system does not response to user, the system shall display informative message about the error. Since the ml algorithms takes huge amount of time the front end side shall not be locked during these processes. There shall be a backup system in order not to lose information of users of the system. The system should be available %90 of time during a year. Moreover there should be a beta environment for future development in order not to cause an unforeseen error. The system shall be implemented such a way that if there will be transportation in the system (such as change the database server or add a new server to the system), the maintainability shall not be broken.

3.2.3 PERFORMANCE

The average response time of the system shall be 200 milliseconds or less. Moreover it shall never be more than 2 second even in extreme cases. It shall be able to work under the load of 1024 concurrent users, and transaction per second shall not be less than 20. All the sql queries shall be written regarding this constraint.

If the system degraded it shall still run the web service. It shall lock the ml engine for a short period of time and show the users that want to run certain tasks, an informative message. All other parts of the system such as demonstrating reports or changing settings shall be available for all users.

The resource utilization shall be monitored by system admins frequently. An infrastructure monitoring tool shall be used for that purpose. Nagios [5] is recommended but not mandatory. The monitoring tool shall send e mail to the admins of the system when the storage or memory is full, critical (more than %75 load), OK state(less than %75 load). It shall also send e-mail on CPU is in critical state. This monitoring tool shall be installed on main web server, notification server, and databases servers. The in-cache memory system does not need to be monitored since it has its own utilization management.

3.2.4 SUPPORTABILITY

The system shall be designed in such a manner that supportability is considered. To run the backend side a process management tool shall be used. Since the server side coded in nodejs pm2 (Advanced, production process manager for Node.js) is highly recommended.

Every component of the system shall keep meaningful of that show the state of the system. Log file shall include date and time and system log files and error log files shall be separated if possible for the sake of simplicity in maintainability.

3.2.5 SECURITY

The system shall be designed concerning security issues. The password of the user shall not be seen on the user screen. (It should be demonstrated as dots instead). The password of the users shall be kept encrypted.

4. DATA MODEL AND DESCRIPTION

4.1 DATA OBJECT

Since the website is designed in asynchronous pattern and does not fit in basic object oriented design concepts, any class diagram or data dictionary will not be provided in this document. For the ML engine related diagrams and explanations are provided in the coming sections.

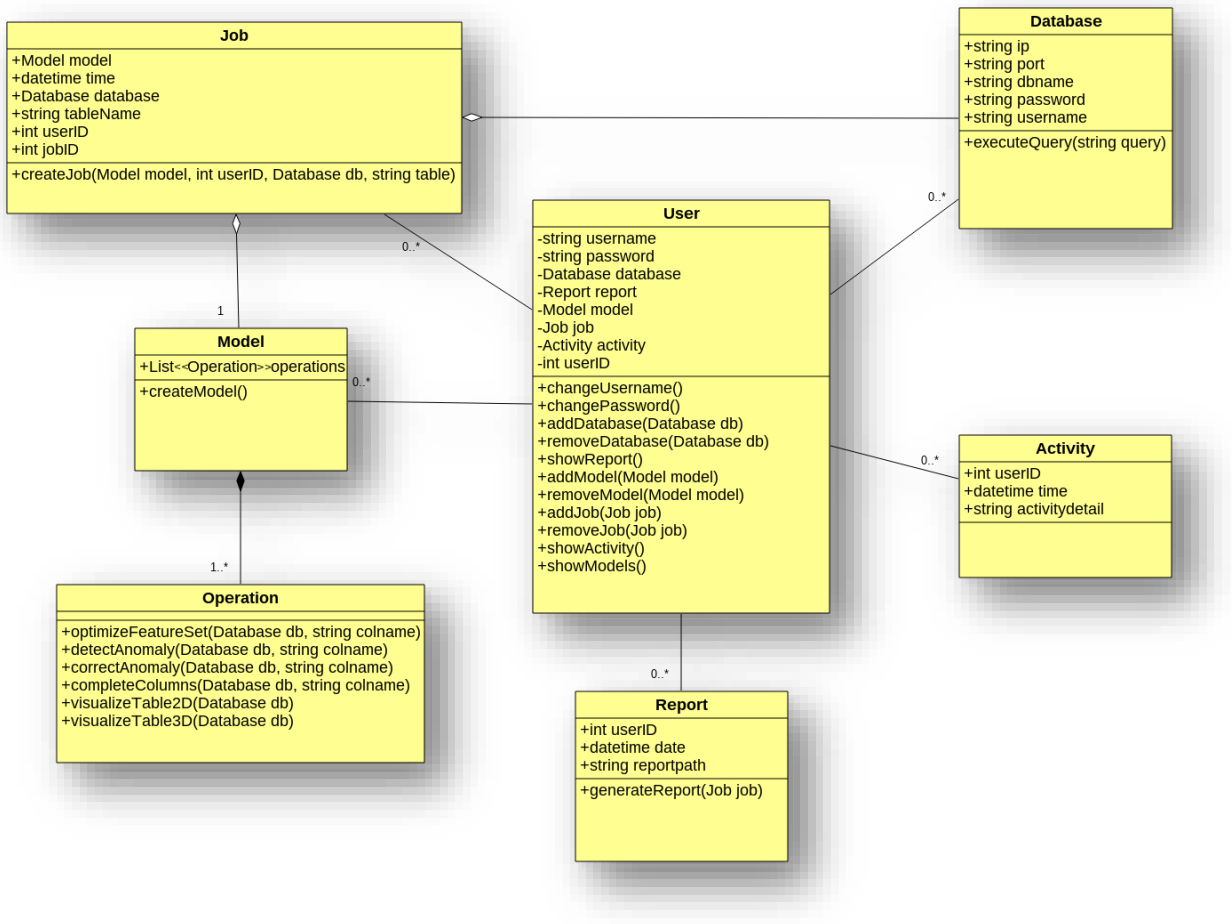


Figure 22: Class Diagram of Vitriol

4.1.1.1 USER

This class represents the member user entity of the system. It has the following attributes: username, password, database, report, model, job, activity. Username and password attributes are primitive data types whereas the remaining attributes are instances of some other classes. It stores the information about the user and his data on the system. This class has related methods to update its fields.

4.1.1.2 DATABASE

This class has the abilities to perform database operations such as connecting, retrieving data etc. It is capable of executing SQL queries on the connected database. It has the attributes of ip, port, dbname, password and username to connect to a database.

4.1.1.3 OPERATION

This class consists of Vitriol's core machine learning and data mining functionalities such as optimizing a feature set (a database table), detection & correction of anomalies, completing tables, visualizing databases for better understanding of the dataset. These methods are utilized for generating models and eventually user defined jobs to perform the desired task.

4.1.1.4 MODEL

This class represents a main construction of Vitriol, namely models. A model consists of multiple sequential operations defining a machine learning model design. This class has the attribute of an operation list, which is operations.

4.1.1.5 JOB

This class represents user defined *job*s. Every task which the user wants to accomplish are stored and processed as jobs in the system. A user task is translated into a model and after that, a job is defined holding that model and also other information about the task including userID and database as a job. This class has the following attributes: model, time, database, tableName, userID and jobID. jobID is used to identify a unique job.

4.1.1.6 REPORT

Report class represents the after operation reports and their generation. The class has the following attributes: userID, data, reportPath. A call to the generateReport(Job job) generates a report in an address in the memory and stores in the object's related field.

4.1.1.7 ACTIVITY

This class represents logs of users' past activities. It has the attributes of userID, time and activitydetail to store a user's a specific activity on the system for future checks.

4.1.2 DATA DICTIONARY

Attribute	Definition
username(User)	It is a string defining the unique username for user to log into the system.
password(User)	It is a string defining the user password for user to log into the system
database	It is the database of the user. For detailed information please refer to section 4.1.1.2
model	It is the model object of the user. For detailed information please refer to 4.1.1.4
job	It is the job object of the user. For detailed information please refer to 4.1.1.5
activity	It is the activity object of the user. For detailed information please refer to 4.1.1.7
userID	It is a string defining the unique id of the user
time	It is a Datetime object defining the time of the job
tableName	It is a string defining the name of the table that the job will be executed.
ip	It is a string defining the ip address of the database
port	It is a string defining the port number of the database
dbname	It is a string that defines the name of the database
password(Database)	It is the password credential of the database
username(Database)	It is the username credential for the database
operations	It is a list consisting of operations that the modal includes. For detailed information please refer to 4.1.1.3
userID(Activity)	It is a string defining the user id that owns the activity object.

time	It is a Datetime object defining the execution time of activity.
activityDetail	It is a string defining the details about the activity.
userID(Reports)	It is a string defining the user id that owns the report object
date(Report)	It is a Datetime object defining the creation date of report.
reportpath	It is a string defining the path of the report in storage

Table 22: Description of the Data Model

Method	**Description**
changeUsername	It is method that changes the username .
changePassword	It is method that changes the user password.
addDatabase	It is a method that add a new database to user's object
showReport	It is a method that shows the reports to the user.
addModel	It is a method that adds the new model to the models of new object.
removeModel	It is a method that removes the model
showActivity	It is method that shows the activities of the user's object.
createJob	It is method that creates the new job.
executeQuery	It is method that executes to given query given database
createModel	It is method that creates a new model in job object
generateReport	It is method that generates the new report to the user
createActivity	It is method that creates the new activities.
optimizeFeatureSet	It is method that optimizes the feature set
detectAnomaly	It is method that detects the anomaly in data
correctAnomaly	It is method that correct anomaly in data
visualizeTable2D	It is method that demonstrates the table in 2D dotplot
visualizeTable3D	It is method that demonstrates the table in 3D dotplot

Table 23: Description of the methods

4.2 ENTITY RELATIONSHIP MODEL

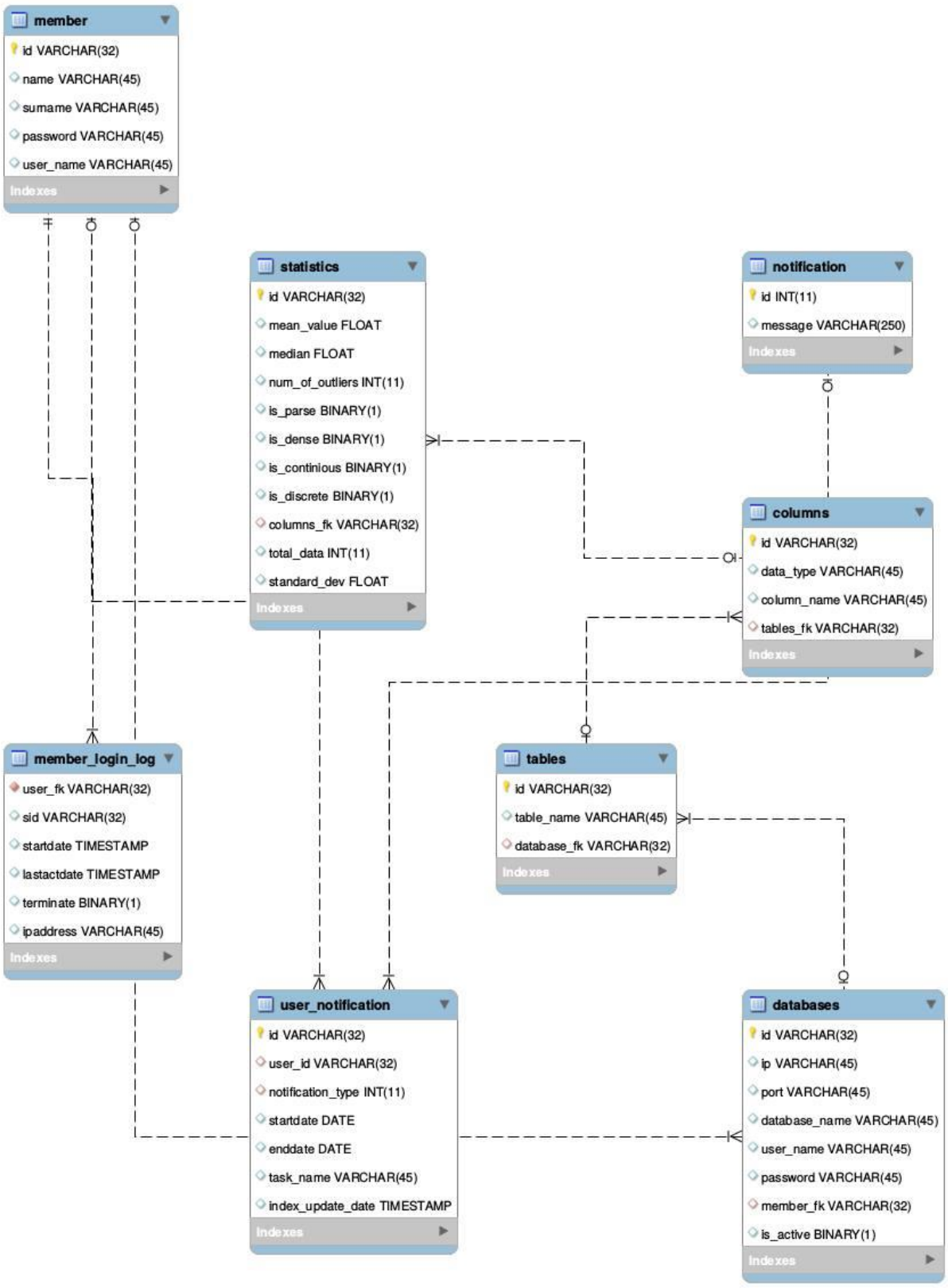


Figure 23: Entity Relationship Diagram

For all entity object except notification primary keys are chosen as 32 bit unique uids because of security issues. Notification table will have very few records, so primary key field for notification is just chosen as integer. The database of the system is designed in such a manner that no information is kept more than once. The tables are connected to each other with foreign keys to keep track of information without replicating the data.

4.2.1 MEMBER ENTITY

Member entity defines the user of the system. It has field's id, name, surname, and password and user name. Password of the users are kept under md5 encryption for the sake of security. User name for each user is unique.

4.2.2 MEMBER LOGIN LOG ENTITY

Member login log entity is the record of the users in terms of sessions. It keeps the information that when a user logs into system and how much does it lasts. User_fk defines the member id, sid defines the specific id for the related session, startdate is the exact time by the session starts. Last act date is the time that the user takes action in the system, Ip address is the Ip of the user and terminate is the information about whether the session is terminated by the server or not(i.e. a multiple login with the same username)

4.2.3 DATABASES ENTITY

Databases entity is simply the records of the databases that the users connected to the vitriol. Ip, port, database_name, user_name and password are nothing but the credentials of the database of the user. member_fk defines the member id, that the database belongs to and is_active shows whether the database is active or not.

4.2.4 TABLES ENTITY

Tables just demonstrate the tables in each database from the databases entity. Table name field is simply defines the name of the table and database_fk demonstrates which database does the table belong to.

4.2.5 COLUMNS ENTITY

Columns demonstrate the columns of each tables. The data_type field is for the type of the column such as integer varchar. Column name is the name of the column and tables_fk defines which table does the column belongs to.

4.2.6 STATISTICS ENTITY

Statistics entity stores the information about the statistics related to the columns. Mean value is the field that the mean value for the column kept. Median is for median and standard_dev is for standard deviation. Is_sparse, is_dense, is_continious is_discrete fields are for the sparse-dense and discrete-continuous information for the column. Also number of the data in the column and number of outliers are kept. Finally columns_fk is for the column id that the statistics belongs to.

4.2.7 NOTIFICATION ENTITY

Notification is for the categorize the notification of the system. At first 4 types of notifications are planned. "These are the process that was started in 'date' is finished", "You are not logged into the system since 'date'", "A new feature is added to the system, would you like to try" and "The system will be unavailable between 'date-date'".

4.2.8 USER NOTIFICATION ENTITY

User notification is for the information that is pushed to the client as notification. It references the notification table for the type of the notification and complete the missing parts with the fields startdate, enddate and task_name. The index_update_date is for the trigger function which keeps the last update time (insert or update) of the row.

5. REFERENCES

[1] IEEE Guide for Software Requirements Specifications," in IEEE Std 830-1984 ,pp.1-26, Feb. 10 1984, doi: 10.1109/IEEESTD.1984.119205,

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=278253&isnumber=6883&tag=1

[2] Nodejs Api Documentation

URL : https://nodejs.org/en/docs/

[3] Pm2 Product Documentation

URL: http://pm2.keymetrics.io/docs/usage/cluster-mode/

[4] Memcached Product Documentation

URL: https://github.com/memcached/memcached/wiki

[5] Nagios Product Documentation

URL: https://www.nagios.org/about/

[6] Centos Product Documentation

URL: https://www.centos.org/about/

[7] Apache Spark Documentation

URL: http://spark.apache.org/docs/latest/